실시간 운영체제 UbiFOS^{TM}에서의 CVM 설계 및 구현

Design and Implementation of CVM on Real-Time Operating System, UbiFOS^{TM}

최찬우, 이철훈
충남대학교 컴퓨터공학과

Choi chan-woo, Lee cheol-hoon
Dept . of Computer Engineering,
Chungnam National Univ.

요약

IT 산업이 빠르게 발전하면서 리소스가 제한된 셋탑박스와 스마트폰 같은 중소형 디바이스의 사용이 비약적으로 증가하는 추세이다. 자바는 플랫폼 독립성(Platform Independency), 보안성(Security), 이동성(Mobility) 등의 장점을 가지고 있기 때문에 안정된 서비스를 제공해야 하는 중소형 디바이스들에게 중요한 핵심 소프트웨어 플랫폼이 되어가고 있다. 이러한 디바이스에서 자바 애플리케이션을 실행하기 위해서는 자바가상머신(Java Virtual Machine)이 필요하다. C 가상 머신(Classic Virtual Machine : CVM)은 리소스가 제한된 임베디드 디바이스를 위해 고안된 자바가상머신이다. 본 논문에서는 실시간 운영체제 UbiFOS^{TM}상에 CDC(Connected Device Configuration) 에서 정의하고 있는 CVM을 설계 및 구현하였다.

Abstract

Having been speedy development of the IT industry, devices such as set-top box and smart phone are used in the broad filed. Because Java has merits that are platform independency, security and mobility, that is important software platform to offer stable services in the small device. This needs JVM(Java Virtual Machine) to execute Java application in the small device. CVM(Classic Virtual Machine) which is the kind of JVM is designed for embedded device to have limited resources. In this paper, CVM which is defined by CDC has designed and implemented on the Real-Time Operating System, UbiFOS$^{TM.}$

I. 서 론

IT 산업이 발전하면서 리소스가 제한된 임베디드 디바이스는 빠르게 발전하고 다양한 기능들을 요구한다. 임베디드 디바이스는 일반 데스크탑 PC 가 아닌 리소스가 제한된 디바이스이다. 이러한 제약 사항 때문에 프로그램은 디바이스에 의존적이며, 따라서 하나의 프로그램이라 해도 각각의 디바이스에 맞추어 수정해야 한다. 이러한 문제점을 해결하기 위해서 플랫폼(Platform)에 독립적인 자바 플랫폼(Java Platform) 기술이 많이 사용되고 있다. 플랫폼 독립성을 가진 자바가상머신(Java Virtual Machine) 기술을 사용하여 다양한 디바이스에 하나의 자바 응용 프로그램을 실행시켜 필요한 기능을 구현할 수 있다. 이처럼 중소형 디바이스에서 자바 응용 프로그램의 실행을 위해서 자바가상머신이 필요하다. J2ME(Java 2 Micro Edition)에 포함된 CDC(Connected Device Configuration)는 리소스가 제한된 디바이스를 위한 CVM(Classic Virtual Machine)이라는 자바가상머신을 정의하고 있다. 본 논문에서는 중소형 디바이스에서 다양한 기능을 제공해주는 CDC에 정의된 CVM을 실시간 운영체제 UbiFOS^{TM} 상에 설계 및 구현한 내용을 기술한다.

본 논문의 구성은, 2 장에서는 관련 연구로서 CVM의 기반 운영체제인 실시간 운영체제 UbiFOS^{TM} 와 CDC에 대한 개념을 기술하며, 3 장에서는 본 논문에서 설계 및 구현한 실시간 운영체제 UbiFOS^{TM} 에서의 CVM에 대해 기술하며, 4 장에서는 테스트 환경 및 결과를 기술한다. 마지막으로, 5 장에서는 결론 및 향후 연구과제에 대해서 기술한다.

본 논문은 국방과학연구소의 임무통제장비 다중화 관리기법 연구과제의 지원으로 수행된 것임.

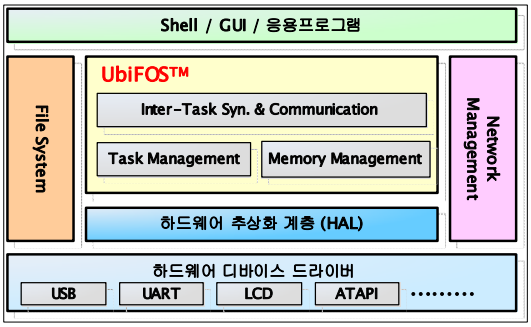
II. 관련연구

CVM 운영 기반인 실시간 운영체제 UbiFOS^{TM} 에 대해서 기술하며, CDC의 개념 및 개발 목적을 기술하며 마지막으로, CDC의 함수 구현에 필요한 POSIX 에 대해 기술한다.

1. 실시간 운영체제 UbiFOS^{TM}

본 논문에서 CVM을 구현하기 위해 기반으로 사용한 UbiFOS^{TM} 는 우선순위 기반 선점형 멀티 쓰레드(Multi-thread)

실시간 운영체제이다. 즉 실시간 운영체제 커널과 응용 프로그램이 통합되어 하나의 큰 프로그램으로 동작하는 구조로써 공통의 메모리 영역을 자유롭게 접근할 수 있고 크기는 약 24KByte 정도이다.



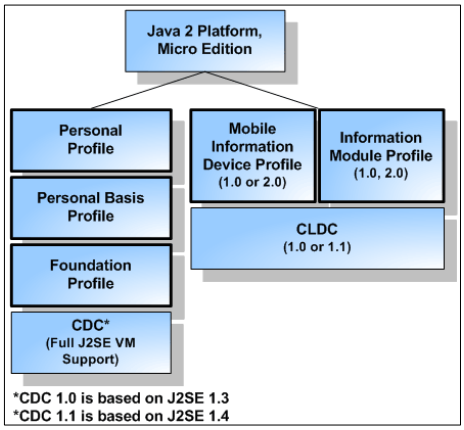
▶▶ 그림 1. UbiFOS™ 의 기능 블록 다이어그램

그림 1. 은 UbiFOS™ 의 기능을 블록 다이어그램으로 나타낸 것으로, 운영체제에서 제공되는 기능은 태스크 관리, 태스크간 동기화 / 통신, 동적 메모리 관리 등이 있다.

시스템 메모리의 힙(Heap) 영역을 관리하기 위해 힙 스토리지 매니저를 사용하며, 힙 영역에서 동적 메모리의 할당은 MK_GetMemory()를 통해 이루어진다[1].

2. CDC(Connetcted Device Configuration)

CDC는 셋탑박스나 스마트폰과 같은 리소스가 제한된 디바이스상에서 자바 응용 프로그램의 실행을 위한 CVM(C Virtual Machine)을 정의한다. CDC 는 J2SE(Java 2 Standard Edition)의 코어 라이브러리 및 클래스 로딩 기능을 포함하며, 중소형 디바이스에서 사용되기 위해서 J2SE의 필요한 코어 클래스 라이브러리의 인터페이스를 디바이스에 맞게 수정하고, 불필요한 라이브러리 클래스는 삭제되었다. CDC는 ROM에서 2MByte 와 RAM에서 2MByte 의 메모리 공간이 필요하다. 이러한 CDC 의 주 목적은 리소스가 제한된 임베디드 디바이스에 J2SE 기술을 적용시키는 것이다[2,3].



▶▶ 그림 2. J2ME 구조

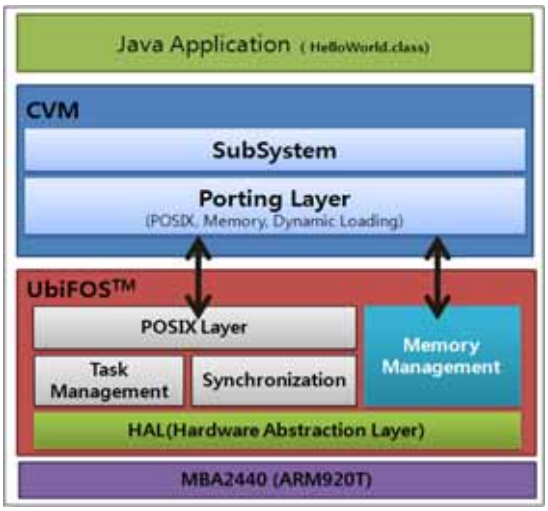
3. POSIX (Portable Operating System Interface for Computer Environment)

유닉스 운영체제에 기반을 두고 있는 일련의 표준 운영체제 인터페이스이다. 표준화에 관한 필요성은, 호환성이 있는 프로그램을 개발하기 원하는 데에서 기인했다.

Ⅲ. CVM 설계 및 구현

1. 설계시 고려사항

CVM 은 새로운 타겟 플랫폼과 운영체제에 포팅을 쉽게 하기 위해서 포팅 라이브러리와 POSIX 함수를 기반으로 하는 Porting Layer를 지원한다. Porting Layer 는 멀티 쓰레드 지원, 메모리 관리 기능을 지원하기 위한 랩퍼(Wrapper) 함수들로 구성되는데, 이러한 랩퍼 함수의 내용을 구현하기 위해서는 UbiFOS™ 에서 POSIX 표준 함수의 지원이 필요하다. 또한, CVM 은 리눅스의 동적 메모리 할당 함수를 사용하기 때문에, 이를 UbiFOS™ 의 메모리 관리 모듈을 사용하여 대체한다.

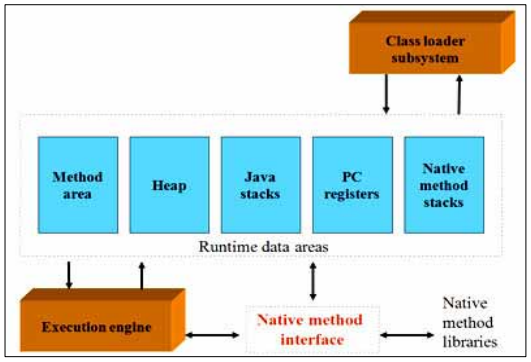


▶▶ 그림 3. 실시간운영체제 UbiFOS™와 CVM 구조

2. CVM의 구현사항

2.1 CVM의 세부 기능

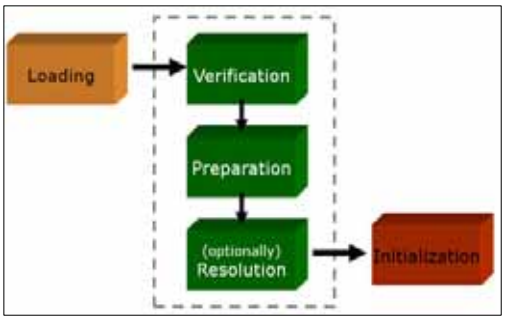
CVM은 그림 4처럼 클래스 로더부분(Class Loader Subsystem)과 클래스 파일을 얻어 와서 가상머신의 실행상태로 만들어주는 클래스 런타임 데이터 영역(Runtime Data Areas), 그리고 바이트 코드의 인스트럭션을 수행하는 실행엔진(Execution Engine)으로 세분화한다.



▶▶ 그림 4. CVM의 구조

2.1.1 클래스 로더 시스템(Class Loader System)

클래스 로더 시스템은 클래스 파일을 얻어와서 가상머신의 실행 상태로 만들어주는 부분으로, 클래스 변수를 저장하기 위해 메모리를 할당하고 초기화하며, 심볼릭 레퍼런스의 Resolution을 돕는다. 이러한 과정은 로딩, 링킹, 초기화 과정을 통해 수행된다.



▶▶ 그림 5. 클래스 로더 시스템 구조

로딩은 특정한 이름을 사용해 클래스나 인터페이스의 바이너리 형태를 찾는 과정으로 세 개의 기본적인 과정으로 구성된다.

- 타입을 표현하는 이진데이터의 스트림을 생성
- 이진데이타의 스트림을 메소드 영역안에서 내부 데이터구조로 파싱
- 타입을 표현하는 java.lang.Class 클래스의 인스턴스 생성 및 내부데이터 구조의 인터페이스의 역할을 담당

링킹은 verification, preparation, resolution 의 3 단계로 수행된다. CVM 에서의 verification 은 단순히 바이트코드를 순차적으로 검증하며 Off-device pre-verification 과 스택 맵을 이용한 런타임 검증이 있다. Preparation 은 클래스 변수를 위한 메모리, 실행중인 프로그램의 수행을 향상시키기 위해 데이터 구조를 위한 메모리(클래스 메소드를 위한 데이터를 포인트하는 메소드 테이블)을 할당한다. Resolution 과정은 런타임시 Constant Pool 에 있는 심볼릭 레퍼런스로부터 동적으로 구체적인 값을 결정하는 방식이다.

초기화 과정은 클래스 변수에 적절한 초기값을 세팅 시키는 작업으로 자바 코드에서 적당한 초기값은 class variable initializer 나 static initializer 에 의해 기술되고, 타입의 class variable initializer 와 static initializer 은 자바 컴파일러에 의해 하나의 특별한 메소드 (<clinit>())에 위치한다.

2.1.2 런타임 데이터 영역(Runtime Data Areas)

런타임 영역은 프로그램이 실행되는 동안 메모리를 구성하는 부분으로 메소드, 힙, 자바 스택, PC 레지스터, 네이티브 메소드 스택으로 구성되며, 자바가상머신의 인스턴스들은 하나의 메소드 영역과 힙영역을 가지며, 이러한 런타임 데이터 영역은 쓰레드(Thread)간 서로 공유한다.

2.1.3 실행 엔진(Execution Engine)

자바가상머신의 구현의 핵심부분으로 실행엔진의 수행은 명령어 집합(Instruction Set)으로 구성된다. 각각의 인스트럭션 명세서에는 해당하는 명령어에 따른 동작을 자세하게 정의한다. 그리고 메소드의 바이트코드 스트림(Bytecode stream)은 자바가상머신을 수행하기 위한 명령 순서로 실행 엔진(Execution Engine)에서 바이트코드를 한 번에 하나씩 수행한다.

2.2 CVM과 UbiFOS™의 연결

CVM 은 Porting Layer에 포함된 함수(그림 6)를 사용하여 멀티 쓰레드, 상호배제, 동기화의 기능을 지원한다. 이 함수들의 내용은 운영체제의 POSIX 표준 함수들을 이용하여 구현하였다. 이러한 CVM 의 요구사항을 위해서 UbiFOS™ 에 POSIX 표준 함수(그림 7)를 구현하였다.

```
// 쓰레드(Thread) 함수
CVMBool POSIXthreadCreate
        (CVMThreadID *tid,
        CVMSize stackSize, CVMInt32 priority,
        void (*func)(void *), void *arg);
CVMThreadID * POSIXthreadGetSelf();
// Mutex 함수
void POSIXmutexLock(...);
void POSIXmutexUnlock(...);
// 동기화(Synchronization) 함수
int POSIXcondvarWait(...);
void POSIXcondvarNotify(...);
```

▶▶ 그림 6. CVM 의 Porting Layer에 정의된 랩퍼(Wrapper)함수

```
// 쓰레드(Thread) 함수
int pthread_create
        (pthread_t *thread,
        const pthread_attr_t *attr,
        void *(*start)(void *), void *arg);
pthread_t pthread_self();
// Mutex 함수
int pthread_mutex_lock(...);
int pthread_mutex_unlock(...);
// 동기화(Synchronization) 함수
int pthread_cond_wait(...);
int pthread_cond_signal(...);
```

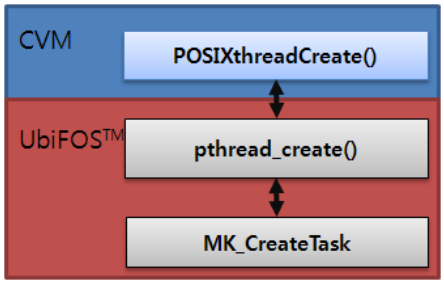
▶▶ 그림 7. UbiFOS™에 구현된 POSIX 표준 함수

실시간 운영체제 UbiFOS™에서 구현한 POSIX 표준 함수를 사용하여 CVM 의 Porting Layer에 포함된 함수를 구현할 때 서로 연관되는 함수들의 상관 관계를 표 1에 나타내었으며, 본 논문에서는 POSIXthreadCreate() 함수의 구현만을 설명한다.

[표 1] CVM 의 랩퍼 함수와 UbiFOSTM 의 POSIX 함수의 상관 관계

CVM	UbiFOS™
POSIXthreadCreate()	pthread_create()
POSIXthreadGetSelf()	pthread_self()
POSIXmutexLock()	pthread_mutex_lock()
POSIXmutexUnlock()	pthread_mutex_unlock()
POSIXcondvarWait()	pthread_cond_wait()
POSIXcondvarNotify()	pthread_cond_signal()

Porting Layer 에 포함된 POSIXthreadCreate() 함수는 멀티 쓰레드를 지원하기 위해서 쓰레드를 생성하는 함수이다. 이 함수의 실제 기능은 UbiFOS™ 의 태스크 관리(Task Management) 함수를 호출하게 되는데, 그때 호출되는 함수는 MK_CreateTask() 함수이다. 그러나 MK_CreateTask() 함수는 플랫폼에 의존적이기 때문에 POSIXthreadCreate()에 의해서 직접 호출되지 않고, 그림 8 에 나타낸 바와 같이 POSIX Layer 를 중간 계층으로 하여 연결한다.



▶▶ 그림 8. 쓰레드 생성시 호출되는 함수의 관계

CVM 은 동적 메모리 할당을 위해서 “stdlib.h” 헤더파일에 정의된 malloc(), calloc() 와 같은 동적 메모리 할당 함수를 사용한다. 이러한 함수들은 UbiFOS™ 의 메모리 관리 방법과 상이하기 때문에 UbiFOS™ 에서 구현된 메모리 관리 모듈로 대체하여 구현한다. 표 2 에서 동적 메모리 할당 함수를 대체하는 UbiFOS™에서 구현된 함수를 나타낸다.

[표 2] 동적 메모리 할당 함수

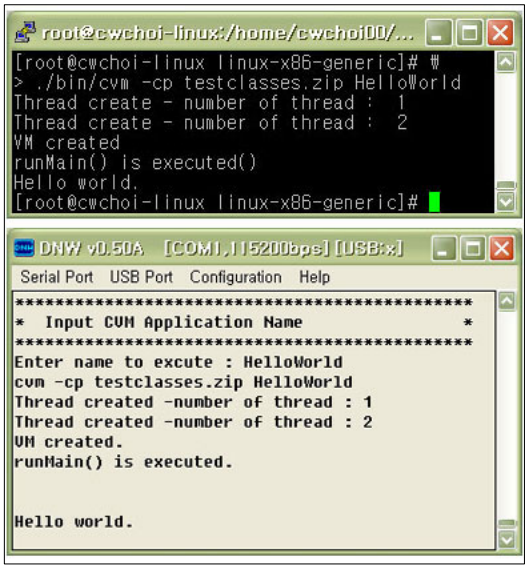
CVM	UbiFOS™
malloc()	CvmMalloc()
calloc()	

Ⅳ. 테스트 환경 및 결과

본 논문에서는 ARM920T 기반의 S3C2440 32-bit RISC Micro Processor 를 사용한 MBA2440 보드 상에 ADS v1.2(ARM Developer Suite ver 1.2) 개발환경을 사용하여 실시간 운영체제 UbiFOS™ 에 CVM을 포팅하여 테스트하였다.

그림 9 는 HelloWorld.java 소스 파일을 컴파일 하여 생성된 HelloWorld 클래스 파일을 리눅스에 포팅된 CVM에서의 실행 화면과 실시간 운영체제 UbiFOS™ 에 포팅된 CVM 을 기본 실행 환경으로 하여 실행 한 결과를 비교한 것이다.

HelloWorld 클래스 파일은 클래스 로더 시스템에 의해서 클래스 파일이 로딩되고, 해당 클래스에 맞는 런타임 데이터 영역이 생성되고 최종적으로 실행 엔진에 의해서 클래스의 바이트코드가 인터프리팅(Interpreting)되면서 실행된다.



▶▶ 그림 9. 리눅스상의 CVM 에서 HelloWorld 클래스 실행화면(위), UbiFOSTM 상의 CVM에서 HelloWorld 실행화면(아래)

V. 결론 및 향후 연구 과제

본 논문에서는 리소스가 제한된 다비이스에 자바 응용 프로그램을 실행시키기 위해서, CDC 에서 정의하고 있는 자바가상머신인 CVM 을 실시간 운영체제 UbiFOS^{TM} 상에서 설계 및 구현하였다.

향후 연구과제로는 CVM 의 다양한 기능을 지원하기 위한 추가적인 기능, 네트워크 기능을 제공하기 위한 Foundation Profile 을 구현해야 하며, 그래픽 유저 인터페이스 기능을 위한 Personal Basis Profile 또는 Personal Profile 을 구현해야 한다. 또한 추가적인 옵션 Profile 을 구현함으로써 CVM 의 추가적인 기능을 지원 할 수 있는 연구가 필요하다.

▌참 고 문 헌▐

[1] Aijisystems, "Embedded hardware & software solution, UbiFOS(Ubiquitous Flexible Real-Time Operating Systems) & MCU platforms".

[2] Sun Microsystems, "CDC (Connected Device Configuration) Runtime guide".

[3] Sun Microsystems, "CDC : JAVATM Platform Technology For Connected Devices, JavaTM Platform, Micro Edition, White Paper", June 2005.

[4] Sun Microsystems, "Inside the JAVA2 Virtual Machine second edition".

[5] 백대현, 정명조, 안희중, 박희상, 이철훈, "임베디드 시스템을 위한 가상머신 분석 및 설계", 한국정보처리학회, Vol. 9, No. 2 (상), pp.543-546, 2002.11.

[6] 유용선, 성영락, 이철훈, "Design of the Java Virtual Machine for Mobile devices", 한국정보과학회, Vol. 30. No. 2(IIII), pp.523-525, 2003.10.